*DOI:*
10.1016/j.actaastro.2020.01.020

*Document Version*
Peer reviewed version

This is the Author's Accepted Manuscript version of the article: Elad Denenberg (2020). Satellite Closest Approach Calculation Through Chebyshev Proxy Polynomials. Acta Astronautica. Accepted for publication on 15 January 2020.

# Satellite Closest Approach Calculation Through Chebyshev Proxy Polynomials

Elad Denenberg

*King's College London*
*Bush House, 30 Aldwych, WC2B 4BG, London, UK*

**Abstract**

Calculating the Time of Closest Approach (TCA) and the minimal distance between two orbiting objects is a vital step in space debris collision risk assessment and avoidance maneuver computation. Currently, two methods are commonly used to compute the TCA and the corresponding minimal distance: propagating both objects with a small time-step and approximating the derivative of a distance function using cubic polynomials. Lately using the Surrogate-based Optimization (SBO) search was also suggested. Propagating is too slow; estimating is usually not accurate enough, and the SBO is considered a good compromise. This paper proposes a novel method based on Chebyshev Proxy Polynomials for the TCA computation problem. This method is designed to allow fast computation and high accuracy; this is demonstrated in several examples, and shown to be slightly slower than the approximation method and have accuracy competitive to those of the SBO method.

*Keywords:* Space Situational Awareness, Spectral Analysis, Time of Closest Approach

*Author's Accepted Manuscript version of the article: Elad Denenberg (2020). Satellite Closest Approach Calculation Through Chebyshev Proxy Polynomials. Acta Astronautica. Accepted for publication on 15 January 2020.

*Email address:* eladden@gmail.com

## 1. Introduction

Finding the Time of Closest Approach (TCA), and the minimum distance between satellites at that time is a crucial step in many space operations, such as service satellites or self-assembling satellites. In these missions, the TCA computation is used in planning a trajectory.

The computation of TCA and the associated distance are also a central part of situational awareness. When an operator or an autonomous spacecraft wishes to check the plausibility of a collision accurately, one of the first steps performed is computing the TCA between the satellite and the threat, and the distance between them at that time [1, pp. 908-926].

When used in situational awareness Inaccuracies in the computation of TCA and the related distance are crucial as they could mean wrongly identifying threats, or worse, not identifying threats as such. The accuracy of the and the related distance depends upon the satellite and debris state estimation, but often the TCA computation itself introduces additional errors that may change the classification of a threat.

Traditionally there were two methods of finding the TCA and the satellite distance at that time: The first, propagating both the objects' trajectories with a very small time-step. The second, called Alfano Negron Close Approach Software (ANCAS) [2], is fitting a proxy polynomial to the distance's derivative and approximating the roots. The first method is costly. Therefore it is not practical to implement on-board an autonomous satellite. Propagating with a small time-step is also not useful when a large amount of computation is required, such as when computing the entire catalog collision probability or a very large cluster of satellites. The second method, though computationally cheap, is not very accurate, and might lead to unreliable results, i.e., claiming the distance to be larger than it is.

Recently a Surrogate-based Optimization (SBO) method was suggested based on ANCAS, which refines the approximations to achieve better accuracy [3]. This method is considered a good compromise between the expensive, exhaus-

tive search of propagating with a small time-step, and ANCAS. However, it still requires a considerable amount of computation power. Thus, even though it is fast and accurate, and would be useful in the computation of many objects conjunction probability, it is still not a very practical tool for on-board satellite implementation.

The space engineering community focus is rapidly shifting from large, expensive satellites to a multitude of cheaper satellites. These small satellites could be a formation or cluster of satellites such as SAMSON [4], TechSat21 [5], PRISMA [6]. Recently initiatives have started for even larger constellation groups such as the StarLink [7] and OneWeb [8]. These small satellites may have reduced computational capabilities. Being a multitude, they would require autonomous capabilities of situational awareness and perhaps decision-making as the operational load of controlling many satellites may become infeasible. These autonomous capabilities require enabling the satellite to accurately estimate risks, which, in turn, requires an accurate way of computing the distance and time of the closest approach.

This work suggests implementing a novel spectral analysis method used in root-finding to facilitate both efficient and accurate TCA and minimal distance computations.

Spectral analysis and Chebyshev methods have been used in several Aerospace applications for quite some time. Between these applications are orbit propagation [9], trajectory optimization [10], and optimal control [11]. For further, more extensive reading regarding Chebyshev polynomials and aerospace Bai's thesis, [12] is recommended.

The method proposed here, named Conjunction Assessment Through Chebyshev Polynomials (CATCH), is an application of the Chebyshev polynomial root-finding method for the fast and accurate computation of TCA and the minimal distance at the conjunction. The proposed method uses state of the art combination of numerical methods with linear algebra. As a result, CATCH requires roughly the same amount of object propagations as ANCAS and achieves high accuracy competitive with the SBO method. Also, this method can be eas-

ily implemented on-board.

The rest of the paper is ordered as follows: The next section will contain background: Section 2.1 will discuss the propagator used in this work, Section 2.2 contains a review of existing methods and Section 2.3 will discuss the optimization method from which CATCH stems. Section 3 will detail the mechanism of CATCH and how it is adapted for the TCA and minimal distance computation. Section 5 demonstrates the method's performance.

## 2. Background

### *2.1. Propagator*

The problem we set to solve in this work is finding the following global minimum:

$$J = \min_t \|\boldsymbol{\rho}(t)\| \tag{1}$$

where $\boldsymbol{\rho}$ is the relative position between any two orbiting objects. For this work, we define $\boldsymbol{\rho}$ to be the distance between the satellites' position $\mathbf{r}_s$ and the position of a debris $\mathbf{r}_d$.

$$\boldsymbol{\rho} = \mathbf{r}_s - \mathbf{r}_d \tag{2}$$

The methods discussed in this paper treat the module calculating the two orbiting objects' positions $\mathbf{r}_s$ and $\mathbf{r}_d$ as a "black box": an evaluation that is triggered when needed, and can be used with any propagator.

For the evaluation, SGP4 was used as a propagator. SGP4 is a common propagator [13],[1, pp. 696–706], which belongs to the Simplified General Perturbations (SGP) models. It incorporates zonals up to J4 and the atmospheric drag. SGP4 is an analytical propagator; therefore, it does not integrate a solution by using Gauss variational equations (GVE) or Two-Body Problem (TBP). Instead, it uses several approximations to estimate the position and velocity of an orbiting object. The SGP models work with a particular format of orbital elements called Two Line Elements (TLE).

The TLE are a set of variables describing an object's orbit. They were developed for the SGP series of propagators, and are based on the Kozai mean

elements [14]. The TLE contain the following orbital information: Spacecraft name and catalog number, eccentricity $e$, mean motion ($n$), Right Ascension of the Ascending Node (RAAN) $\Omega$, inclination $i_c$, argument of perigee $\omega$, mean anomaly($M$), the first and second derivatives of mean motion ($\dot{n}$ and $\ddot{n}$, not used in SGP4) and a drag parameter $B^*$. The North American Aerospace Defense Command (NORAD) provides information about orbiting objects in terms of TLE through their website [15].

TLE and SGP4 have two primary advantages. The first is its availability. Information about orbiting objects is openly accessible online. SGP4 was implemented for several computational platforms. Vallado's SGP4 implementation for Matlab® was used in this work [16]. The second advantage of SGP4, being an analytical propagator, is its computational cost, it is simple to implement and can be used on-board.

The major disadvantage of both the propagator and the format is the accuracy. A generated TLE may contain inaccuracies of up to a kilometer in position and m/sec in speed [17]. The SGP4 propagation error is also quite large, tens of kilometers after a few days [1, pp. 696–706],[18]. However, since this work treated the propagator as a black box, the contribution does not depend on the propagator. Thus, it was decided to use SGP4 while demonstrating the number of calls required to achieve the desired accuracy.

### *2.2. Related Work on Time of Closest Approach*

There are about 22,300 objects currently being tracked in space. Satellite operators wish to know the risk each of those poses to their craft. In order to reduce the number of computations, filters are used on the catalog that contains all the objects. Uninteresting objects are removed from it before trying to compute the risk.

Uninteresting objects are those that clearly do not pose any threat as they are too far. For instance, a satellite in Geostationary Orbit (GEO) will not pose a threat to a satellite in Low Earth Orbit (LEO). A common and easily computed filter is Hoot's geometrical [19]: if the difference between the maximal perigee

and the minimal apogee is larger than a safety distance $D$, then the orbits will never come closer than that distance.

$$q = \max\left(a_s\left(1-e_s\right), a_d\left(1-e_d\right)\right) \tag{3a}$$

$$Q = \min\left(a_s\left(1+e_s\right), a_d\left(1+e_d\right)\right) \tag{3b}$$

$$q - Q \geq D \tag{3c}$$

where $(\cdot)_s$ denotes the satellite, $(\cdot)_d$ the debris, and $D$ is a safety distance defined by the operator.

However, such geometrical filters would still leave thousands of objects in the catalog, with which the minimal distance to the operated satellite needs to be computed. Therefore the filters usually consist of different stages (often named sieves), which introduce computations of growing complexity that eventually result in a much smaller list of possible risks [19, 20, 21, 22, 23].

Since these filters are required to estimate the distance, they often output an estimate of each object's TCA and its corresponding minimal distance to the operated satellite. These estimates are inaccurate and are used as an indicator of the need to compute the actual minimal distance. The computation is often done by integrating the relative distance with a small time step. This integration is a computationally expensive process that could not be implemented as a means of filtering or on-board autonomous spacecraft.

Other than integrating the relative distance over time, two methods for computing the TCA exist: the common ANCAS and the recent SBO based algorithm.

ANCAS is an old, yet a still relevant method for quickly estimating the TCA. Due to its speed, it is often used as a sieve in filters [23]. It starts by defining a distance function:

$$f\left(t\right) = \boldsymbol{\rho}\left(t\right) \cdot \boldsymbol{\rho}\left(t\right) \tag{4}$$

where $\boldsymbol{\rho}$ is the euclidean distance between the two objects position: $\boldsymbol{\rho} = \mathbf{r}_1 - \mathbf{r}_2$

It follows that:

$$\dot{f}(t) = 2\dot{\boldsymbol{\rho}} \cdot \boldsymbol{\rho} \tag{5}$$

$$\ddot{f}(t) = 2\left(\ddot{\boldsymbol{\rho}} \cdot \boldsymbol{\rho} + \dot{\boldsymbol{\rho}} \cdot \dot{\boldsymbol{\rho}}\right) \tag{6}$$

Therefore, a local minimum would be found when

$$\dot{f}(t^*) = 0 \tag{7a}$$

$$\ddot{f}(t^*) > 0 \tag{7b}$$

ANCAS fits a piecewise continuous cubic spline to a sample set of $\dot{f}$.

$$C(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3,\ 0 \leq \tau \leq 1 \tag{8}$$

The TCA is approximated by finding the roots $t^*$ at which all $C(\tau) = 0$. The distance at the TCA $\|\boldsymbol{\rho}(t^*)\|$ is calculated using three quintic polynomials fitted between $\boldsymbol{\rho}(t_i)$ and $\boldsymbol{\rho}(t_{i+1})$. The TCA is the time in which the estimated distance is minimal.

Though it depends on the choice of $\Delta t$, it can be said that ANCAS is a fast method for computing the TCA, but it lacks accuracy. [3] has suggested implementing a Surrogate-based Optimization (SBO) search in order to improve this accuracy, and allow the user better control over the distance's accuracy, rather than tuning the time-step.

ANCAS uses proxy functions, functions that can be easily optimized, to approximate the TCA and minimal distance. A surrogate function is a proxy that is refitted in an iterative process to achieve better fitting and more accurate results.

SBO is an optimization method that has lately seen much work in the field of aerospace design [24, 25]. The search algorithm is:

1. Fit a surrogate proxy to the function being optimized
2. Find surrogate minima
3. Acquire candidate points for sampling based on the previous step
4. Sample the optimized function at candidate points

5. Repeat until convergence

The suggested SBO algorithm for the computation of TCA uses the roots of $C(\tau)$, $\tau^*$, as candidate points, and fits a new cubic using them and the closest known 3 points. The process is then repeated until the sample, and the estimation diverges less than a required distance tolerance $\epsilon_f$ or a defined time tolerance $\epsilon_t$.

Though the SBO method is accurate, it still is a compromise on computational power and may require a large amount of re-sampling.

### *2.3. Chebyshev Proxy Polynomials Root-finding*

The method proposed in this work is based on ideas from the works of [26]. This is a global optimization method which has implementations readily available in numerous platforms (i.e. MatLab [27], C++ [28], Python [29]).

The method is a root finder: fitting a given function with a Chebyshev Proxy Polynomials (CPP) then using linear algebra methods to find the polynomials' roots. ANCAS uses polynomials as proxies. However, unlike ANCAS, the method presented here uses CPP as a proxy rather than a simple cubic.

Linear algebra tools have been used in the generalization of ANCAS in [3]. The tool described in [3] is based on a method that extracts the roots of a multivariate polynomial by calculating the Macaulay matrix's eigenvalues [30]. Similarly,[26] presents a method for extracting the roots by calculating the companion matrix's eigenvalues. The companion matrix is a sparse matrix that can be composed by the polynomial coefficients. Details on the companion matrix can be found in Section 2.3.3

The root-finding method of [26] is composed of three steps:

1. Fitting the CPP
2. Checking tolerance convergence and raising the polynomial order or bisecting the search space if needed
3. Extracting the roots

The next subsections will detail each step.

*2.3.1. Fitting the Polynomial*

Let $g(x)$ be a function to which we wish to find the roots, defined on the interval $x \in [a, b]$. An approximation Chebyshev polynomial would be:

$$g(x) \approx g_N(x) = \sum_{j=0}^{N} a_j T_j \left( \frac{2x - (b+a)}{b-a} \right) \tag{9}$$

where $N$ is the polynomial order and $T_j$ is

$$T_j(x) = \cos(j \arccos(x)) \tag{10}$$

Unlike ANCAS where the time points sampled for the fitting were uniformly spread over the interval, to fit a CPP, we sample at the Chebyshev-Gauss-Lobatto nodes:

$$x_j = \frac{b-a}{2} \cos\left(\pi \frac{j}{N}\right) + \frac{b+a}{2} \tag{11}$$

where $j = 0 \dots N$ is the point index, and $N$ the polynomial's order.

The $(N+1) \times (N+1)$ interpolation matrix $\mathcal{I}$ is defined as

$$\mathcal{I}_{j,k} = \frac{2}{p_j p_k N} \cos\left(j\pi \frac{k}{N}\right) \tag{12}$$

where $p_j$ is the following function

$$p_j = \begin{cases} 2 & j = 0, N \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

The coefficients $a_j$ are computed from the following matrix-vector multiplication:

$$\mathbf{a} = \mathcal{I} g_N \tag{14}$$

where $\mathbf{a}$ is a vector containing all coefficients. in other words:

$$a_j = \sum_{k=0}^{N} \mathcal{I}_{j,k} g_k \tag{15}$$

Where $j = 0 \dots N$ and $g_k = g(x_k)$.

*2.3.2. Checking for Tolerance Convergence*

The CPP order determines the number of points that will be sampled and, therefore, the estimator's accuracy. The higher the order, the closer $g_N(x)$ is to $g(x)$. Here we detail the process of selecting the order in a way that ensures the error does not exceed a required $\epsilon$.

Note that any $M$ Chebyshev nodes contain the previous $\frac{M}{2}$ nodes [31]. Also, note that $|T_q| \leq 1$ since it is a cosine function. Therefore, the maximal approximation error between an approximation and that of an order that is doubled can be computed by fitting a CPP to the difference between them:

$$\delta_{2N} = |g_{2N} - g_N| = \sum_{0}^{2N} \tilde{a}_j T j \tag{16}$$

where $\tilde{a}_j$ are the approximation's coefficients $\delta_{2N}$. The error of $f_{2N}$ is bound by

$$E \leq \sum_{0}^{2N} |\tilde{a}_j| \tag{17}$$

The error defined here drops geometrically with the doubling of the order. Therefore, if the error between $g_N$ and $g_{2N}$ is under the desired threshold $\epsilon$, then the error between $g_{2N}$ and $g$ would not be significantly higher than that threshold.

Thus to find the required order, one starts by fitting a polynomial of order 2, then a polynomial of order 4. If the error is higher than $\epsilon$, the order is doubled to 8, and the process thus repeats until convergence.

*2.3.3. Extracting the Roots*

ANCAS used a cubic polynomial because it has an analytic solution to real roots finding problem. Here, we use a CPP of an arbitrary order $N$. The CPP roots can be found by calculating the companion matrix's eigenvalues.

We build the $N \times N$ companion matrix $\mathbf{A}$ from the CPP coefficients.

$$\mathbf{A}_{j,k} = \begin{cases} \delta_{2,k} & j = 1, \quad k = 1 \dots N \\ \frac{1}{2}\left(\delta_{j,k+1} + \delta_{j,k-1}\right) & j = 2 \dots N-1, \quad k = 1 \dots N \\ -\frac{a_{k-1}}{2a_N} + \frac{1}{2}\delta_{N-1,k} & j = N \quad k = 1 \dots N \end{cases} \tag{18}$$

where $\delta_{q,r}$ is a binary function that is 1 when both indices are equal, and 0 otherwise:

$$\delta = \begin{cases} 1 & q = r \\ 0 & otherwise \end{cases} \tag{19}$$

In other words the matrix would look like this:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & \dots & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & \dots & 0 & \frac{1}{2} & 0 & \frac{1}{2} & \\ -\frac{a_0}{2a_N} & -\frac{a_1}{2a_N} & & \dots & & -\frac{a_{N-2}}{2a_N} + \frac{1}{2} & -\frac{a_{N-1}}{2a_N} & \end{bmatrix} \tag{20}$$

The companion matrix eigenvalues $\mathbf{e} = \text{eig}(\mathbf{A})$ are the roots of $g_N$ mapped to the interval $[-1, 1]$. The roots of $g_N$ can be rescaled back to the interval $[a, b]$ thus:

$$x^* = \frac{b+a}{2} + e_i \frac{b-a}{2} \tag{21}$$

where $x^*$ is a root in interval $[a, b]$ and $e_i$ is the $i$th eigenvalue of $\mathbf{A}$.

Calculating the eigenvalues of matrices can be computationally expensive, depending on the matrix size and implementation. However, if the maximal order $N_{max}$ is small enough, the computation can be kept within a practical computational burden. Implementations of eigenvalue calculations are abundant. For C++, for instance, one can use the boost linear algebra library [32], Eigen [33], or Armadillo [34]. Such libraries exist for many different platforms. Matlab was used for testing in this paper.

*2.3.4. Practical Drawbacks*

The root-finding method described in this section suffers from several practical drawbacks, and [26] suggest different methods of overcoming them.

The first is non-smoothness of the approximated function or a function that is non-analytical. In the general case, the singularities can be "taken out" by sectioning the search space in a way that would remove them. In some cases, singularities may give rise to complex roots with small complex elements.

The second group of practical problems that may arise is with bounds. Unbounded functions need to be mapped to a finite interval. Functions with a dynamic range (very large numbers on some of the defined range while very small in other areas) may also cause difficulties with machine precision.

Though the original article addresses these drawbacks and finds possible solutions for them, we will not bring them here, as they are not relevant to our problem. A relevant drawback, however, is the computational cost of finding the eigenvalues. The cost of the QZ decomposition required for the eigenvalue computation of a matrix of size $N$ is proportional to $N^3$. To avoid the need for finding the eigenvalue of an ever-growing matrix, [26] suggests bisecting the interval. This procedure is described in Algorithm 1.

First attempt to fit the polynomial over the interval $[a, b]$ with a growing order $N$. If $N$ reaches a maximal order $N_{max}$, divide the interval into two intervals: $\mathcal{I} = \left\{ \left[a, \frac{a+b}{2}\right], \left[\frac{a+b}{2}, b\right] \right\}$, and repeat the fitting process for each section. When the error converged to the desired value for all sections, build the companion matrices for all the sections. The companion matrices' eigenvalues will be the estimated function's roots.

Indeed this bisecting process is economical since it ensures the decomposed matrices are kept smaller than $N_{max} \times N_{max}$. However, a large number of sections could lead to a large number of points sampled in order to fit the CPP; If the evaluation of the approximated function is costly, this could lead to a slow search process.

There is, therefore, a trade-off between choosing a small $N_{max}$ and a large $N_{max}$. A small $N_{max}$ would allow the fast computation of the roots, but might lead to a large number of sections and thus require many samples. A large $N_{max}$ would lead to a small number of sections, but extracting the roots can become computationally expensive.

**Algorithm 1:** The Chebyshev Proxy Polynomials (CPP) Rootfinding

**Input:** Interval $[lb, ub]$, function $f(x)$, desired maximal error $E_{max}$, maximal order $N_{max}$

**Output:** $\mathcal{R}$ a set containing the roots of $f(x)$

1 $N = 4$;
2 $\mathcal{I} = \{[lb, ub]\}$;
3 **for** *each interval* $[a, b]$ *in* $\mathcal{I}$ **do**
4  **while** $N \leq N_{max}$ **do**
5  Fit $N$ order CPP according to Eq. (9)–(15);
6  Calculate error $E$ according to Eq. (17);
7  **if** $E > E_{max}$ **then**
8  $N = 2 * N$;
9  **end**
10  **if** $N >= N_{max}$ **then**
11  Replace current interval $[a, b]$ in $\mathcal{I}$ with $\left\{\left[a, \frac{a+b}{2}\right], \left[\frac{a+b}{2}, b\right]\right\}$ ;
12  Go to Line 3 and repeat to include new sections;
13  **end**
14  **end**
15 **end**
16 **for** *each interval in* $\mathcal{I}$ **do**
17  Compute the companion matrix (Eq. (18));
18  Add the companion matrix eigenvalues to $\mathcal{R}$;
19 **end**

The CPP root-finding method's full workings are depicted in Algorithm 1: First, the algorithm fits a polynomial to the interval (Line 5), then the size of the error is computed. If the error is greater than a given threshold, then the order $N$ is doubled. If $N$ is greater than a given maximum $N_{max}$, then the interval is bisected.

After the piecewise continuous splines were fitted and are within the desired

tolerance, the algorithm will compute the companion matrix and the roots of each piece's interval.

## 3. Conjunction Assessment Through Chebyshev Polynomials (CATCH) Method

The method presented here is an implementation of the CPP root-finding method to the problem of finding the TCA and relevant distance. The novelty is not only in the use of spectral analysis, the CPP root-finding, but also in the adaptation to the specific problem at hand while avoiding the practical pitfalls of the original root-finding algorithm.

The distance function is defined in the same way as in Eq. (4):

$$f(t) = \boldsymbol{\rho}(t) \cdot \boldsymbol{\rho}(t)$$

It is safe to assume this distance between the satellites is a continuous derivable function; thus, a fitted CPP will not possess any imaginary roots. The propagation of each satellite separately may contain critical undefined points, depending on the specific orbital element method, and the propagator. However, the propagation of position and velocity based on a current state is a continuous function. Since no discontinuities are present in each satellite position and velocity, the distance function is continuous as well, and the definition presented here of the scalar multiplication also enforces drivability. This guarantees that identifying critical points and removing them is not required in this case.

Another assumption safely done here is that the search for the TCA is done within a finite horizon. Even if the search for TCA will be for the entire satellite's lifetime, that time is finite. Thus, no theoretical problems may arise for a lack of boundaries.

We are left only with the computational drawback that arises due to the method's iterative nature. CATCH overcomes this drawback by making the following three observations: First, within a single orbit, the distance function can have no more than four extrema: two maximum distance and two mini-

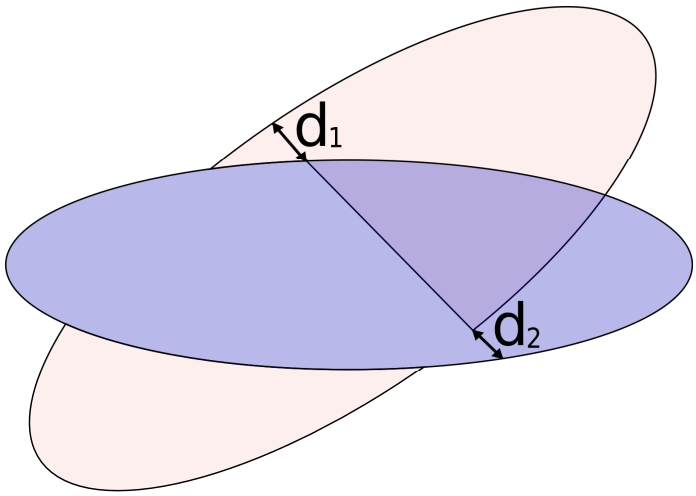



Figure 1: Two Locally Minimal Distances

mum distance because the orbits are ellipses that share the same focal, the two minimal distance points are demonstrated in Fig 1.

The second observation is that the distance function is recurrent with a governing frequency that is determined by the two objects' orbital period.

Therefore, the following can be said: If $T_a$ is the orbital period of one object and $T_b$ The orbital period of the second, then let $\Gamma$ be the time interval:

$$\Gamma \leq T_{min} = \min\left(T_a, T_b\right) \tag{22}$$

There will be no more than 4 extrema in the interval $\Gamma$.

In other words, we can divide the entire search space to $\Gamma$ time intervals. In every interval $\Gamma$ we seek the following $t_k$ time points of the extrema:

$$\dot{f}\left(t_k\right) = 0 \tag{23}$$

$t_k \in [0, \Gamma]$ and $k \in \mathbb{N}$, $k \leq 4$.

Thus, It is not necessary to bisect the search space iteratively as described in Section 2.3, since the governing period is known, it is sufficient to divide the test period $t_{max}$ into $\Gamma$ intervals smaller than that period and seek the local minima in them.

The third observation is that most objects for which the TCA calculation will be done, are orbiting in similar regimes (i.e., LEO with a small eccentricity orbit, or GEO orbit). Finding the $\Gamma$ and $N$, which provide satisfactory results for some orbits, will prove useful in most orbits. Therefore, reiterating the parameter selection is not required. We have found that $N = 16$ and $\Gamma = \frac{T_{min}}{2}$ were sufficient in all observed cases.

Relying on this last observation is not trivial: though about 90% of objects are either near-circular or with moderate eccentricity ($0.01 < e \leq 0.1$) [35, pp. 16-18], after basic geometrical filtering the catalog we may remain with a large number of elliptic orbits (24% in the example we tested in the evaluation). Experiments show that though the performance of CATCH is lowered in High Elliptical Orbit (HEO), it is still acceptable in terms of safe operations. This point is demonstrated and discussed further in the evaluation section.

In the next section, the process of selecting the parameters will be demonstrated. Once the parameters governing the error (the interval size $\Gamma$ and the order $N$) are decided, the CPP root-finding method is used to compute the TCA without iteration. Next, it is used again to compute the distance at the TCA.

For the distance computation, CATCH fits additional CPP to $\rho_x$, $\rho_y$, and $\rho_z$. Since the same order is used, the same sampled data can be used to fit these CPP.

Let $\mathcal{F}_x$ be the CPP approximation of $\rho_x$ ($\rho_y$ and $\rho_z$ are analogous):

$$\mathcal{F}_x = \sum_{0}^{N} a_j T_j \left( \frac{2t - q\Gamma}{\Gamma} \right) \tag{24}$$

where $q$ is the section number in which the minima is found, i.e. the first interval is $[0, \Gamma]$ the second is $[\Gamma, 2\Gamma]$ and so on until $[(M-1)\,\Gamma, t_{max}]$, $M$ being the total number of sections: round $\left(\frac{t_{max}}{\Gamma}\right)$.

Let $t^*$ be the time of the exremum. The distance at the extremum is estimated using CPP thus:

$$r_{\min} = \sqrt{\mathcal{F}_x^2\left(t^*\right) + \mathcal{F}_y^2\left(t^*\right) + \mathcal{F}_z^2\left(t^*\right)} \tag{25}$$

**Algorithm 2:** CATCH Algorithm

**Input:** œ$_1$,œ$_2$, $t_{max}, \Gamma, N$

**Output:** TCA and $r_{TCA}$ distance at TCA

1 $\mathcal{T}\dashv = [\,]$ ;
2 $r_{TCA} = \inf$;
3 $t_{TCA} = 0$ ;
4 $a = 0$;
5 $b = \Gamma$ ;
6 **while** $b \leq t_{max}$ **do**
7  Fit CPP of order $N$ to $f(t)$ according to Eq. (9)–(15) to $[a, b]$;
8  Compute the companion matrix (Eq. (18));
9  Add the companion matrix eigenvalues to $\mathcal{T}$;
10  Fit CPP of order $N$ to $\rho_x$, $\rho_y$ and $\rho_z$;
11  **for** *each* $t*$ *in* $\mathcal{T}\dashv$ **do**
12  Compute distance $r_{min}$ as per Eq. (25);
13  **if** $r_{min} < r_{TCA}$ **then**
14  $r_{TCA} = r_{min}$;
15  $t_{TCA} = t^*$;
16  **end**
17  **end**
18  $a+ = b$;
19  $b+ = \Gamma$;
20 **end**

The method is summed up in Algorithm 2: The algorithms receives as an input the orbital elements of two orbiting objects œ$_1$,œ$_2$, and a time horizon over which to search $t_{max}$. The search interval, therefore, becomes $[0, t_{max}]$.

The algorithm uses Eq. (9)–(15) to fit a CPP (Line 7). After computing the $N \times N$ companion matrix, the local extrema are extracted from its eigenvalues. Finally, Three additional CPP of order $N$ are fitted to $\rho_x$, $\rho_y$, and $\rho_z$, and the

distance at the found extrema is computed (Line 10). If the distance at the extrema is lower than any previously found, the time is stored as the TCA and the distance as the minimum (Line 13 – 16).

Note that Algorithm 2 returns the TCA and not a list of local minima. This is because the local minima are not stored in memory. It can easily be implemented to return the list of all local minima if these are of interest, and the satellite memory is capable: by replacing Line 13 – 16 to store these points.

## 4. Parameter Selection

The runtime of CATCH depends on several factors: the propagator and the time it takes to sample the points, the number of points sampled, the number of fitted polynomials, and the polynomials' order. The propagator is outside the scope of this work; the other factors are directly influenced by the time interval $\Gamma$ and the order $N$.

The number of times a CPP will be fitted is euqal to the number of intervals $M = \text{round}\left(\frac{t_{max}}{\Gamma}\right)$. Therefore, given a time horizon $t_{max}$ a smaller $\Gamma$ will cause $M$ to be large, and that would cause a larger amount of polynomial fittings.

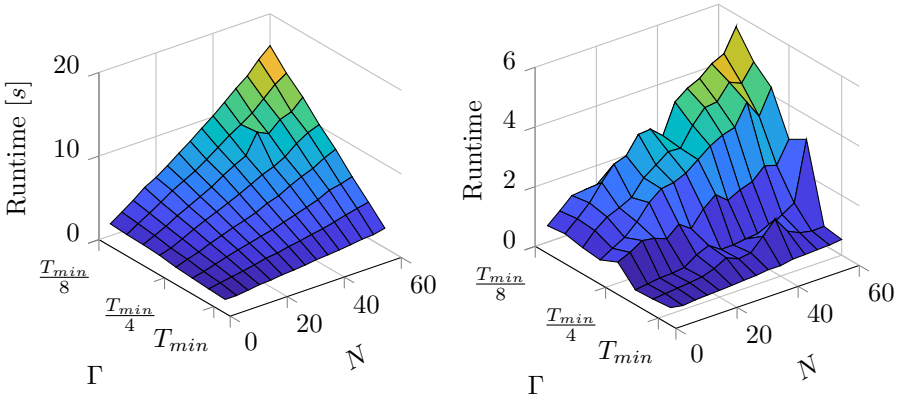
The number of points sampled is the CPP order multiplied by the number of sections $M$:

$$S = M * N \tag{26}$$

where $S$ is the number of sampled points.

The influence of $\Gamma$ and $N$ on the runtime is demonstrated in Fig 2. The figure represents the runtime as a function of $\Gamma$ and $N$ in two typical orbits of LEO (SKYSAT-1 and FENGYUN 1C DEB) and GEO (EXPRESS-AM6 and RADUGA-1 3). These are typical results. The orbits' details are represented in the evaluation section.

As expected, a rise in $N$ and a lowering of $\Gamma$ cause an increase in the runtime. The errors, however, drop exponentially. This can be seen in Fig 3, where the error in time and distance for both these orbits are represented. Again, these results are typical.




(a) LEO

(b) GEO

Figure 2: Runtime in LEO and GEO

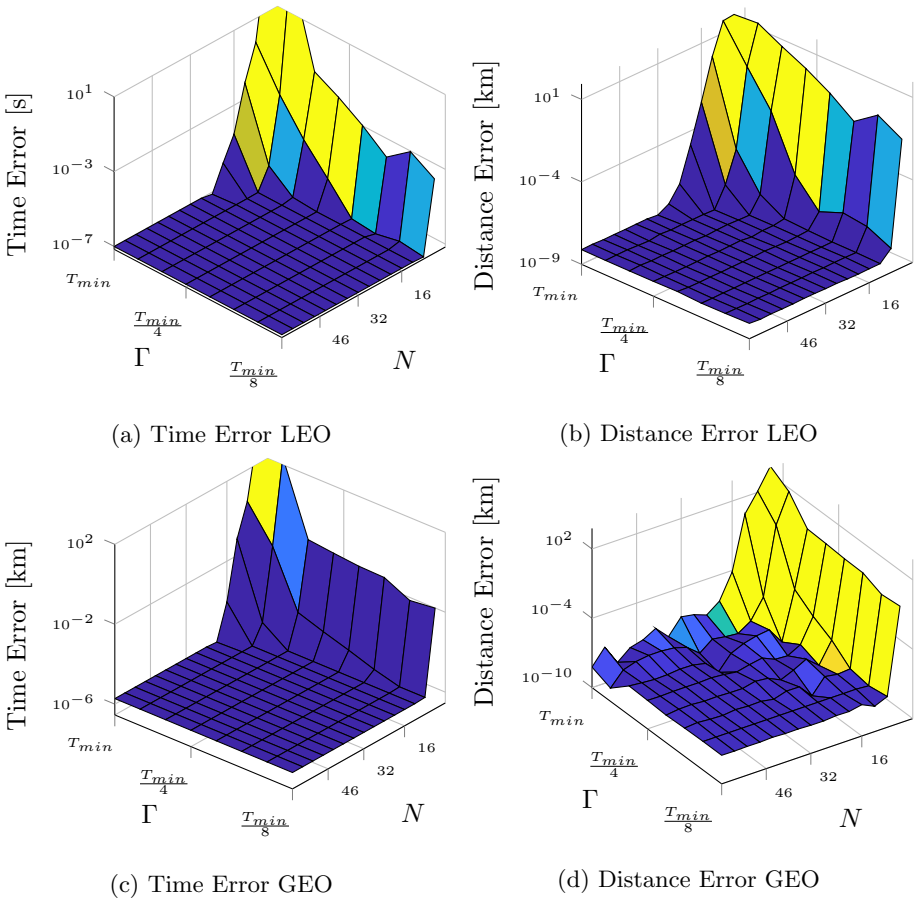
Plotting such graphs of error as a function of $\Gamma$ and $N$ show that $\Gamma = \frac{T_{min}}{2}$ and $N = 16$ are enough to achieve high accuracy, with a very low runtime cost. Using these values provided results with satisfactory accuracy in all the instances we examined. Finding the eigenvalues of a $16 \times 16$ matrix is a computationally simple task and can easily be done on low-performance computers such as may be found on-board a satellite.

## 5. Evaluation

To demonstrate CATCH's performance, we compare it to the performance of ANCAS and SBO-ANCAS. All tests were run on an Intel® i7-8550U CPU 1.80GHz ×8 using Matlab®. The propagator used was Vallados implementation of SGP4 [16]. Five problems were tested: Satellites in LEO, GEO, a satellite in LEO and object inHEO, a debris cloud, and one-on-all (the computation of the entire catalog and a single satellite).

For fairness of comparison, the constant time-step selected for ANCAS in all cases was $\Delta t = {}^{\Gamma}/_{32}$. This ensures that approximately the same amount of samples is used for spline fitting in ANCAS and CATCH.

The SBO-ANCAS started with an initial sampling step of $\Delta t = {}^{\Gamma}/_{32}$. The




(a) Time Error LEO

(b) Distance Error LEO

(c) Time Error GEO

(d) Distance Error GEO

Figure 3: Estimation of Error

distance tolerance requirement was $\epsilon_t = 10^{-5}$s, and the time tolerance $\epsilon_f = 10^{-5}$km in all tests. Recall, these tolerances define when the iterative process of refining the approximation will end.

The examples were taken from the Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space (SOCRATES) website [36].

### *5.1. Low Earth Orbit (LEO)*

Two scenarios of objects in LEO are brought here. The first is that of COSMOS 1607 and FENGYUN 1C DEB. The threat of these two objects colliding was explored in [3]. The TLE of the two objects is given in Table 1. The

Table 1: COSMOS 1607 and FENGYUN 1C DEB Two Line Elements

| **Object** | **Two Line Elements** |
|---|---|
| COSMOS 1607 | 1 15378U 84112A 15194.50416942 -.00000089 00000-0 -28818-6 0 9993 |
| | 2 15378 64.9934 313.9757 0056000 236.4761 194.3519 13.83536818552983 |
| FENGYUN 1C DEB | 1 31570U 99025BZM 15193.80658714 .00004278 00000-0 28637-2 0 9998 |
| | 2 31570 102.6018 188.9155 0192610 30.1541 89.8377 13.93148752415999 |

Table 2: COSMOS 1607 and FENGYUN 1C DEB Results

| **Algorithm** | **Run time (s)** | **Evaluations** | **Surrogate Built** | **Error (km)** |
|---|---|---|---|---|
| ANCAS | 0.682 | 12490 | 3248 | 126.51 |
| SBO-ANCAS | 1.835 | 22434 | 25118 | 1.37e-08 |
| CATCH | 0.807 | 12482 | 1560 | 3.84e-08 |

TCA was expected 4.693 days from COSMOS's epoch, and the minimal given distance was 0.116 km.

The results of the three algorithms: ANCAS, SBO-ANCAS, and CATCH are shown in Fig 4, and further information is brought in Table 2.

The real global minimum that the errors in Table 2 are measured against was computed using Matlab multi-start algorithm from Matlab's global optimization toolbox. This algorithm is slow and found the minimum within minutes. Therefore, it is only brought here as a reference for the real minimum.

In the top part of Fig 4, the distance between the satellites is shown in the blue line. The real minimum is marked with a triangle. The local minima that were found by ANCAS are marked with a plus, SBO-ANCAS computed minima are marked with x's and CATCH global minimum as marked with a square. The $10^4$ near both axes signifies that the numbers shown on the axes are in tens of thousands (i.e., 1.4e4 km on the y-axis).

The bottom part of Fig 4 shows a zoom-in around the real minimum. Notice that the ANCAS estimated minimum distance at the TCA is significantly higher than the real minimum. This may cause the algorithm to classify the conjunction as not threatening, or identify a different local minimum as the TCA. This is

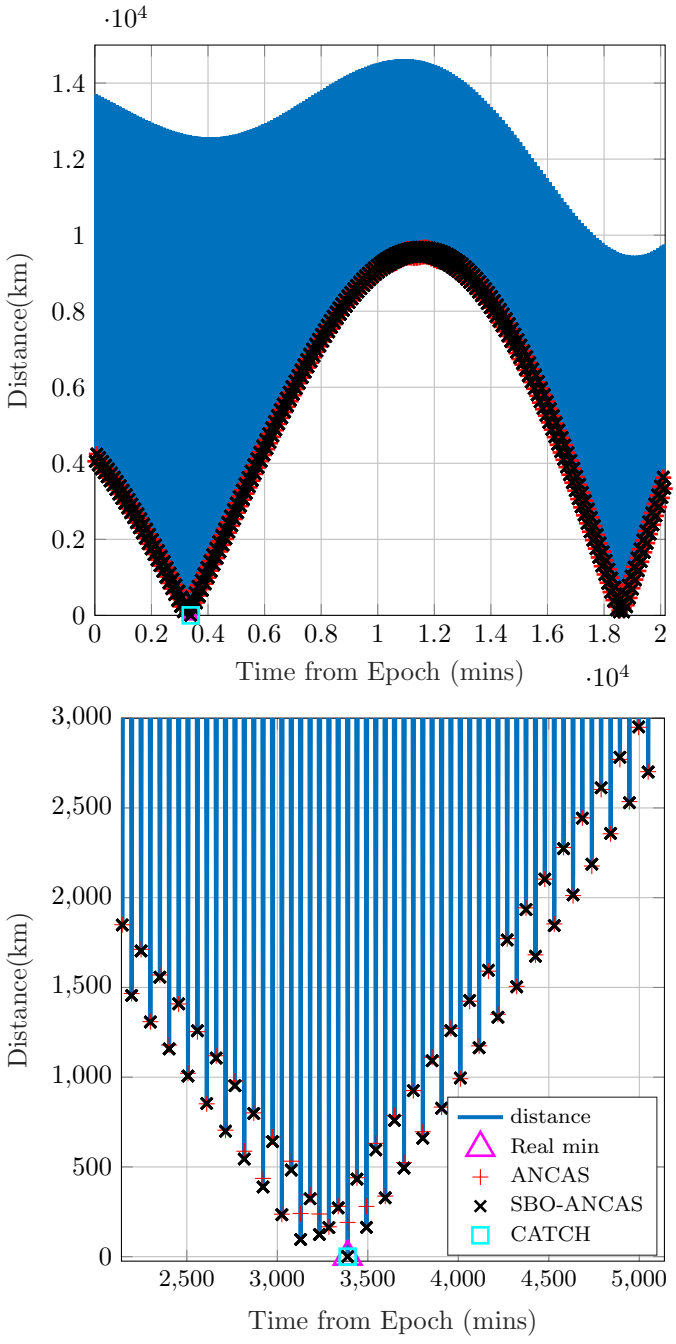



Figure 4: COSMOS 1607 and FENGYUN 1C DEB

seen better in Table 2.

The error in Table 2 is computed as the absolute value of the difference between the algorithm computed minimum and the real minimum. Table 2 shows that the fastest method is ANCAS. However, as expected, and shown in [3], that method is the least accurate. SBO-ANCAS, though far more accurate, is much slower. The accuracy achieved is higher than the given tolerance; this is due to the tight time tolerance.

CATCH runtime is of the same order as that of ANCAS. The number of times CATCH and ANCAS sample the objects' position (the column named "Evaluation") is very close, CATCH fits the polynomial much less. The difference in time is since finding the eigenvalues of a $16 \times 16$ matrix takes much longer than analytically finding the minimum of a cubic polynomial.

The number of polynomials fitted in ANCAS is about twice that of CATCH. One would expect the difference to be more significant since ANCAS fits a polynomial to every four points while CATCH to each 16. The reason the difference in the number of fitted functions is not great is explained by the fact that CATCH fits four surrogates to each interval whileANCAS would fit at most four.

Since $\Gamma$ is half an orbit, it is expected that at least one extrema will be found within each interval, when using CATCH. Thus, for each interval a polynomial is fitted to $f$ as well as to $\rho_x$ , $\rho_y$, and $\rho_z$. ANCAS, in contrast, fits a polynomial between every four points, and there is no guarantee that a minimum exists. Therefore, only if the cubic has a real root and the distance is a local minimum at that root, then does ANCAS fit the three additional quintic polynomials to $\rho_x$, $\rho_y$, and $\rho_z$. Thus, the number of fitted polynomials is double rather than four times that of CATCH.

The second examined case in LEO is that of SKYSAT-1 and FENGYUN 1C DEB. The two-line elements of these objects are brought in Table 3. The TCA computed by SOCRATES was 4.055 days from the SKYSAT-1 epoch, and the given minimal distance was 0.006 km.

The results are shown in Fig 5 and Table 4. As in the previous case, the

Table 3: SKYSAT-1 and FENGYUN 1C DEB Two Line Elements

| Object | Two Line Elements |
|---|---|
| SKYSAT-1 | 1 39418U 13066C 19042.20180787 -.00000231 00000-0 -14135-4 0 9996 |
| | 2 39418 97.6613 126.3667 0023103 57.2848 303.0584 14.98788403285542 |
| FENGYUN 1C DEB | 1 36258U 99025DWK 19042.03235178 .00000157 00000-0 33857-4 0 9997 |
| | 2 36258 99.0353 139.3348 0205784 231.9458 126.3017 14.62433061544647 |

Table 4: SKYSAT-1 and FENGYUN 1C DEB Results

| Algorithm | Run time (s) | Evaluations | Surrogate Built | Error (km) |
|---|---|---|---|---|
| ANCAS | 0.790 | 13438 | 3061 | 28.24 |
| SBO-ANCAS | 1.561 | 20116 | 20837 | 2.24e-7 |
| CATCH | 0.887 | 13442 | 1680 | 1.02e-07 |

upper plot of Fig 5 shows the distance function and the found minima, and the lower plot a zoom in around the real minimum.

It can be seen in Table 4 that CATCH is slower than ANCAS but much faster than SBO-ANCAS. The error is computed from the real minimum found by Matlab's global optimization tool. Here too the accuracy of CATCH was slightly better than that of SBO-ANCAS, and both are smaller than 0.5mm.

### *5.2. Geostationary Orbit (GEO)*

Since GEO satellites orbit at higher altitudes, the governing period $\Gamma$ is larger than those in LEO, and therefore sampling $N$ points per orbit would result in fewer sampled-points per-minute. This may affect ANCAS; however, the SBO algorithm is immune to this phenomenon as it refines its approximation.

CATCH is expected to have the same accuracy in GEO as in LEO. The points are not equally spaced in time; rather, they are positioned in a way that minimizes the fitting error.

The GEO scenario here is a conjunction between EXPRESS-AM6 and RADUGA$-1$ 3. The TLE of the objects are given in Table 5. The minimal distance is expected to be 0.13km, and the TCA is 2.7 days since the EXPRESS-AM6 epoch.

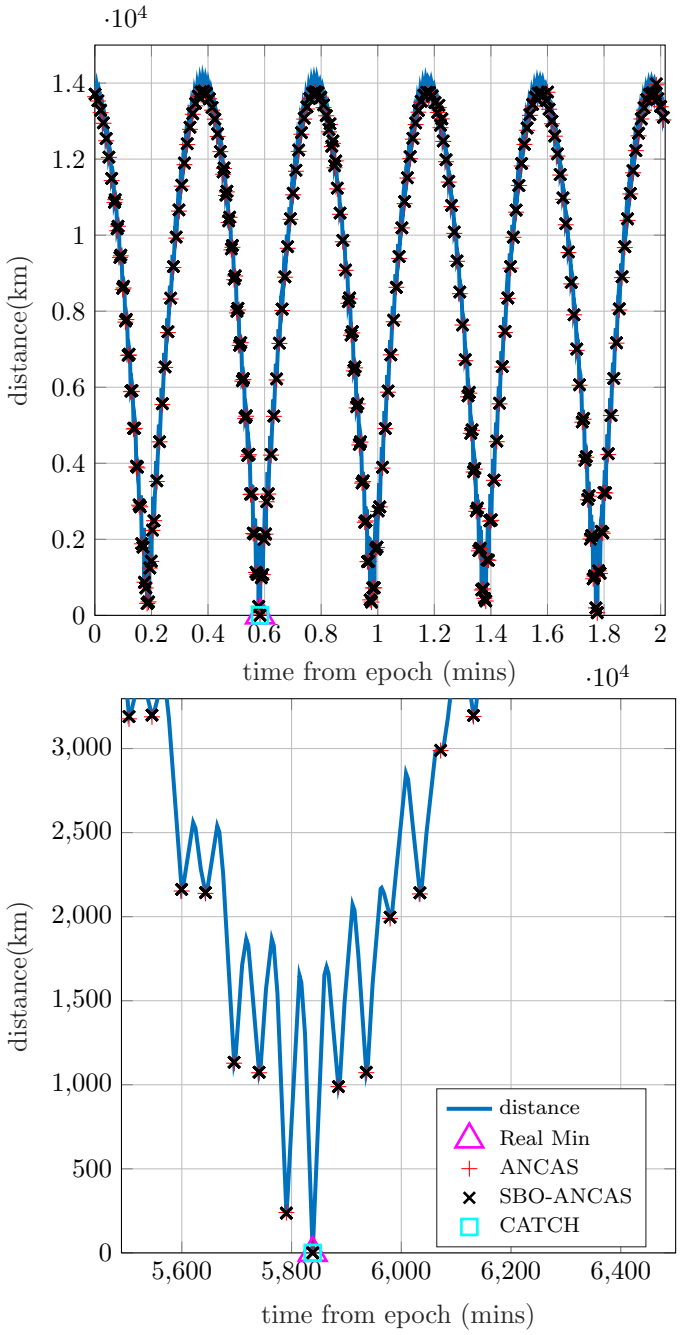



Figure 5: SKYSAT-1 and FENGYUN 1C DEB

Table 5: EXPRESS-AM6 and RADUGA-1 3 Two Line Elements

| Object | Two Line Elements |
|---|---|
| EXPRESS-AM6 | 1 40277U 14064A 18357.91924417 .00000104 00000-0 00000+0 0 9990 |
| | 2 40277 0.0504 207.9211 0000588 97.6225 170.7741 1.00270069 15272 |
| RADUGA-1 3 | 1 22981U 94008A 18357.65221042 .00000072 00000-0 00000+0 0 9997 |
| | 2 22981 14.9263 10.8951 0004005 303.6212 65.7034 1.00241161 91112 |

Table 6: EXPRESS-AM6 and RADUGA-1 3 Results

| Algorithm | Run time (s) | Evaluations | Surrogate Built | Error (km) |
|---|---|---|---|---|
| ANCAS | 0.358 | 900 | 233 | 131.21 |
| SBO-ANCAS | 0.725 | 1762 | 2102 | 6.18e-9 |
| CATCH | 0.374 | 898 | 112 | 3.86-08 |

The results of the conjunction computation of EXPRESS-AM6 and RADUGA-1 3 are shown in Fig 6 and Table 6. The top plot in Fig 6 shows the distance throughout the testing period. The lower plot shows a zoom around the real minimum.

Since it was not accurate, ANCAS has classified the wrong local minimum as the global one and thus was unable to identify the correct TCA and its correlated minimal distance. The accuracy of SBO-ANCAS and CATCH of order $10^{-2}$mm. Indeed, SBO-ANCAS's accuracy is higher by an order of magnitude, but anything under a millimeter is practically null. It has taken SBO-ANCAS double the time to compute the TCA.

CATCH is shown to achieve similar accuracy in GEO as in LEO using the same parameters.

### *5.3. High Elliptical Orbit (HEO)*

As stated previously, though most of the space objects are in near-circular orbits, a large amount is in elliptic orbits. These are usually rocket bodies and other debris that were generated from transfer orbits. The parameters of CATCH were selected while assuming the objects are both in circular orbits.

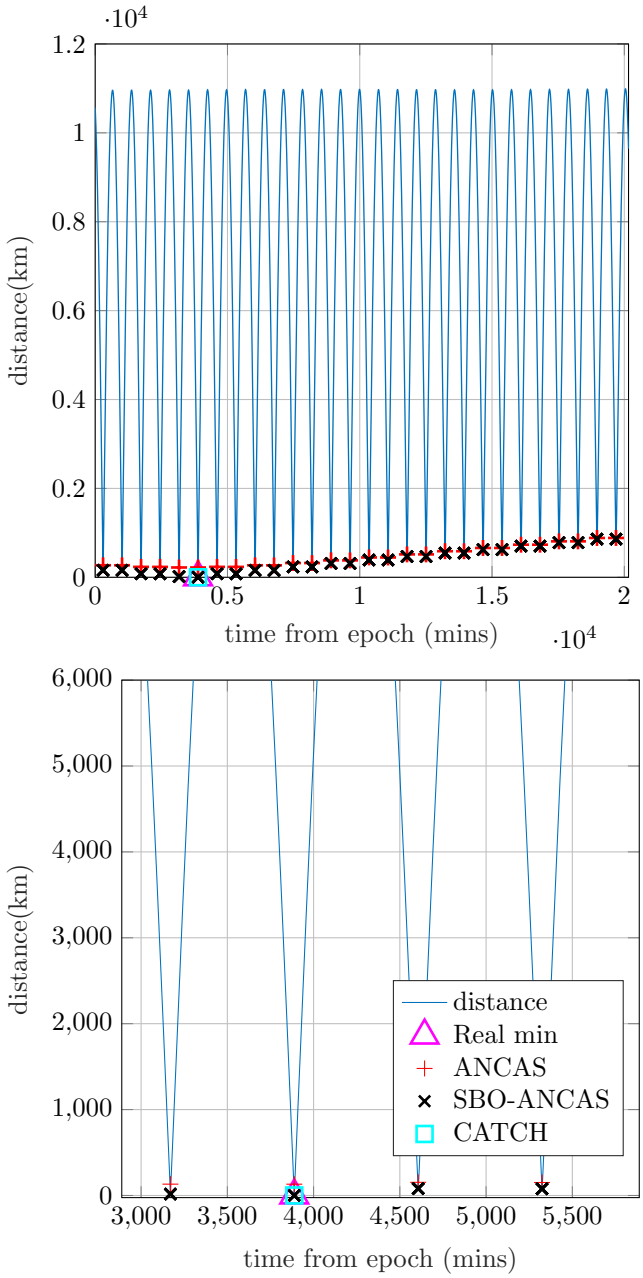



Figure 6: EXPRESS-AM6 and RADUGA-1 3

Table 7: ESSA 1 (OT-3) and ARIANE 2 DEB Two Line Elements

| **Object** | **Two Line Elements** |
|---|---|
| ESSA 1 (OT-3) | 1 01982U 66008A 19281.09103563 .00000000 00000-0 10966-4 0 9994 |
| | 2 01982 97.8261 78.3834 0082923 74.6868 286.3472 14.46902211825154 |
| ARIANE 2 DEB | 1 27595U 88040E 19280.43481983 -.00000015 00000-0 90997-3 0 9992 |
| | 2 27595 7.2727 236.7859 7177087 358.5511 0.0235 2.28760537143592 |

Table 8: ESSA 1 (OT-3) and ARIANE 2 DEB Results

| **Algorithm** | **Run time (s)** | **Evaluations** | **Surrogate Built** | **Error (km)** |
|---|---|---|---|---|
| ANCAS | 1.178 | 12974 | 2162 | 155.57 |
| SBO-ANCAS | 1.941 | 17372 | 4605 | 4.84e-08 |
| CATCH | 1.346 | 12962 | 1620 | 1.44e-5 |

Therefore, we test whether CATCH remains useful when computing the TCA of an object in LEO and another in HEO.

The following scenario is brought here: the conjunction of the ESSA 1 (OT-3) satellite in near-circular orbit and the rocket body debris of ARIANE 2 in an HEO. The two-line elements of these objects are brought in Table 7.

The results are shown in Fig 7 and Table 8. As in the previous examples, the top plot of Fig 7 shows the distance and the found minima. The lower plot is a zoom-in around the real minimum.

It can be seen in Table 8 that, as in the previous examples, CATCH is slightly slower than ANCAS. Here again, ANCAS fails to compute the minimal distance. The accuracy of CATCH here is far lower than that demonstrated in LEO and GEO. However, the accuracy achieved is considered acceptable in practice, and therefore the selected parameters are shown to fit HEO as well.

### *5.4. Debris Cloud and One-on-All*

In this subsection, we will explore the method's performance in case multiple calculations are required. Two cases are explored: A debris cloud and a One-on-All catalog run.

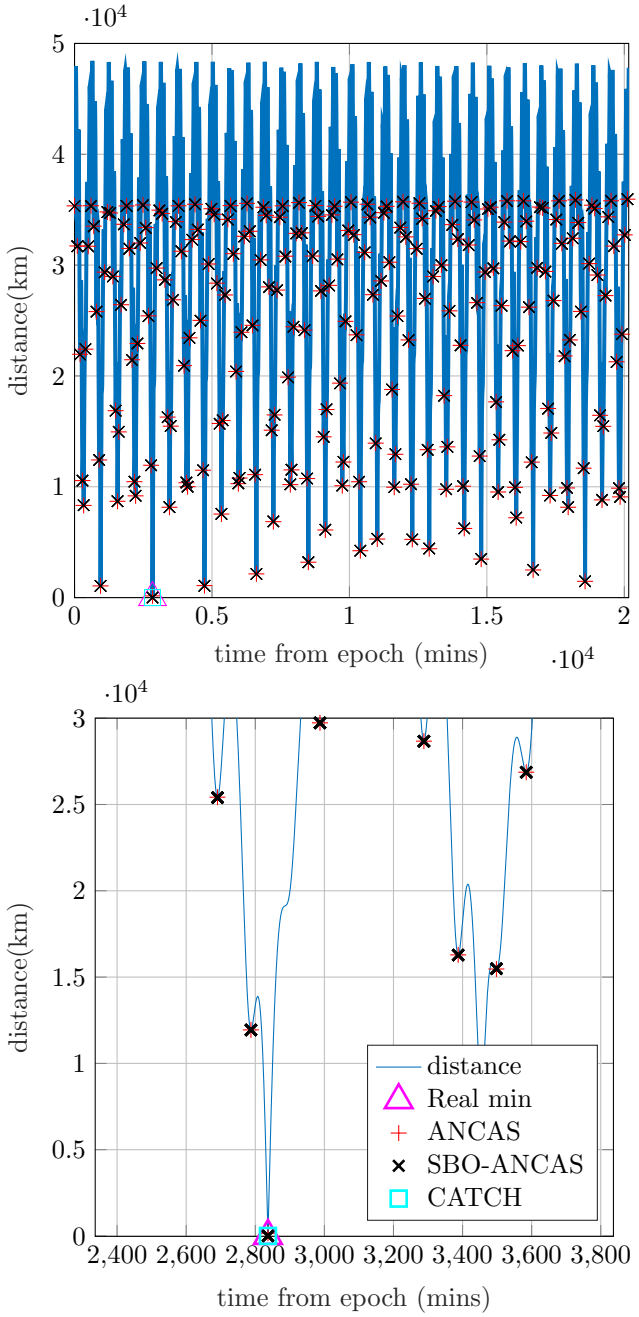



Figure 7: ESSA 1 (OT-3) and ARIANE 2 DEB

Table 9: Debris Cloud

| Algorithm | Run time (s) | Evaluations | Surrogate Built | Error (km) |
|---|---|---|---|---|
| ANCAS | 550.69 | 13,632,000 | 2,885,540 | 6.54 |
| SBO-ANCAS | 931.18 | 18,005,110 | 16,540,060 | 1.41e-07 |
| CATCH | 647.11 | 13,636,000 | 1,704,000 | 7.02e-06 |

*5.4.1. Debris Cloud*

In [3], a scenario of a debris cloud was introduced: a satellite passes through a ring of debris formed by a collision. The 2000 debris objects are scattered uniformly, varying in mean anomaly ($-\pi \leq M \leq \pi$). This scenario is close to that of a cloud of debris in LEO approximately a day after an explosion [35, pp.70-75].

In reality, such a scenario would not require the computation of all debris as filters can be applied. However, here we compute the minimal distance between each debris and a satellite, as this demonstrates further the proposed method's strength and speed.

The results are shown in Table 9. As in the previous examples, SBO-ANCAS runtime was slightly less than double that of ANCAS. While CATCH's runtime is closer to that of ANCAS, the accuracy achieved is 7mm.

These results suggested CATCH may replace ANCAS as a sieve in the various filters. To emphasize this, a demonstration of computing the minimal distance and TCA of a single object with the entire catalog of tracked objects. This is presented in the next subsection.

*5.4.2. One-on-All*

SBO-ANCAS, ANCAS, and CATCH are tested on the highly demanding task of computing the distance between an object and the rest of the tracked objects catalog.

The satellite for which the computation will be done is IRIDIUM 133; its TLE are presented in Table 10. The entire catalog from [15] is first filtered using the geometrical filter described in Eq. (3). After the geometric filtering, 5563

Table 10: Iridium 133 Two Line Elements

| **Two Line Elements** |
|---|
| 1 42955U 17061A 19286.62184192 .00000091 00000-0 25521-4 0 9998 |
| 2 42955 086.3962 324.2863 0002732 094.3868 265.7640 14.34217668105256 |

Table 11: One on All Results

| **Algorithm** | **Run time (min)** | **Evaluations** | **Surrogate Built** |
|---|---|---|---|
| ANCAS | 106.44 | 71,706,081 | 17,832,240 |
| SBO-ANCAS | 244.26 | 120,807,711 | 129,260,155 |
| CATCH | 123.25 | 71,730,613 | 8,964,240 |

objects remain for which the TCA with Iridium 133 is computed.

Out of the 5563 objects 36.78% are in near circular orbit ($e \leq 0.01$), 39.01% are in Moderate Elliptic Orbit (MEO) ($0.01 < e \leq 0.1$). Out of the remaining 24.21% of the orbits, 13.35% are HEO ($0.6 \leq e \leq 0.8$).

The algorithms' runtime results are shown in Table 11. Note, here the runtime results are given in minutes. As expected, CATCH has performed slightly slower than ANCAS.

If the SBO-ANCAS results are used as a baseline for the error, then ANCAS's median time error is 0.0018sec, and 38.7% of the results achieved accuracy of less than $1 \cdot 10^{-3}$sec. CATCH, on the other hand, had a median of $6.64 \cdot 10^{-9}$sec, and 98.77% of the results were accurate to withing $1 \cdot 10^{-3}$sec.

These results imply that with small additional computation cost CATCH achieves accuracy that is far better than ANCAS. If CATCH is used instead of ANCAS as a sieve in the different filters, the accuracy of the estimated TCA and distance will improve by several orders of magnitude, while paying a small price of computational load.

## 6. Conclusions

This paper presented CATCH, a novel method for computing the TCA and corresponding minimal distance between two orbiting objects. The method is based on a CPP root-finding algorithm that employs spectral analysis and linear algebra tools. The root-finding algorithm is adapted to the TCA finding problem by exploiting unique problem characteristics. The resulting method is both fast and accurate.

The parameters governing CATCH's speed and accuracy were discussed, and an example was given to their selection. CATCH was tested using these parameters on scenarios containing objects in LEO, GEO, HEO, as well as a debris cloud scenario and a one-on-all test with the entire catalog. It was demonstrated that CATCH is only slightly slower than ANCAS, yet has an accuracy that is competitive with that of SBO-ANCAS.

Though CATCH does not depend on the selection of orbital elements and propagator, future work should demonstrate CATCH on systems that run different propagators and orbital elements. It is interesting to see analysis such as shown in Section 4 on different systems, including such that use orbital elements containing information about impulsive or continuous thrust maneuvers.

Future work will also include generalizing the root-finding algorithm to reason with multivariate polynomials, allowing the computation of a debris cloud. Additionally, this method should be implemented on-board a satellite or an emulation of such and tested in a mission-like environment.

[1] D. A. Vallado, Fundamentals of Astrodynamics and Applications, 4th Edition, Springer, 2013.

[2] S. Alfano, Determining satellite close approaches, part ii, The Journal of the Astronautical Sciences 42 (2) (1994) 143–152.

[3] E. Denenberg, P. Gurfil, Improvements to time of closest approach calculation, Journal of Guidance, Control, and Dynamics (2016) 1–

13 doi:10.2514/1.g000435.
URL http://dx.doi.org/10.2514/1.g000435

[4] P. Gurfil, J. Herscovitz, M. Pariente, The samson project–cluster flight and geolocation with three autonomous nano-satellites.

[5] R. Burns, C. A. McLaughlin, J. Leitner, M. Martin, Techsat 21: formation design, control, and simulation, in: 2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484), Vol. 7, 2000, pp. 19–25 vol.7. doi:10.1109/AERO.2000.879271.

[6] S. D'Amico, J.-S. Ardaens, S. D. Florio, Autonomous formation flying based on gps — prisma flight results, Acta Astronautica 82 (1) (2013) 69 – 79, 6th International Workshop on Satellite Constellation and Formation Flying. doi:10.1016/j.actaastro.2012.04.033.
URL http://www.sciencedirect.com/science/article/pii/S0094576512001488

[7] D. Heaven, Elon musk's space internet, New Scientist 240 (3203) (2018) 5. doi:10.1016/S0262-4079(18)32060-8.
URL http://www.sciencedirect.com/science/article/pii/S0262407918320608

[8] J. N. Pelton, B. Jacqué, Distributed Internet-Optimized Services via Satellite Constellations, Springer International Publishing, Cham, 2017, pp. 249–269.
URL https://doi.org/10.1007/978-3-319-23386-4_96

[9] O. Montenbruck, E. Gill, Satellite orbits: models, methods and applications, Springer Science & Business Media, 2012, Ch. 3, pp. 73–75.

[10] F. Fahroo, I. M. Ross, Direct trajectory optimization by a chebyshev pseudospectral method, Journal of Guidance, Control, and Dynamics 25 (1) (2002) 160–166.

[11] H. Jaddu, Direct solution of nonlinear optimal control problems using quasilinearization and chebyshev polynomials, Journal of the Franklin Institute 339 (4) (2002) 479 – 498. doi:10.1016/S0016-0032(02)00028-5.
URL http://www.sciencedirect.com/science/article/pii/S0016003202000285

[12] X. Bai, Modified chebyshev-picard iteration methods for solution of initial value and boundary value problems, Ph.D. thesis, Texas A&M University, 400 Bizzell St, College Station, TX 77843, USA (2010).
URL https://oaktrust.library.tamu.edu/bitstream/handle/1969.1/ETD-TAMU-2010-08-8240/BAI-DISSERTATION.pdf

[13] F. R. Hoots, P. W. Schumacher Jr, R. A. Glover, History of analytical orbit modeling in the us space surveillance system, Journal of Guidance, Control, and Dynamics 27 (2) (2004) 174–185.

[14] Y. Kozai, The motion of a close earth satellite, The Astronomical Journal 64 (1959) 367.

[15] North American Aerospace Defense Command, Space track, accessed: 22/10/2019.
URL https://www.space-track.org/#recent

[16] D. Vallado, Fundamentals of astrodynamics and applications - software and data, accessed: 03/12/2018 (Mar 2013).
URL https://www.celestrak.com/software/vallado-sw.php

[17] J. L. Foster Jr, The analytic basis for debris avoidance operations for the international space station, in: Space Debris, Vol. 473, 2001, pp. 441–445.

[18] S. H. Knowles, A comparison of geocentric propagators for operational use, in: AAS/AIAA Astrodynamics Conference, Halifax, Nova Scotia Canada, 1995, pp. 95–429.

[19] F. R. Hoots, L. L. Crawford, R. L. Roehrich, An analytic method to determine future close approaches between satellites, Celestial Mechanics 33 (2)

(1984) 143–158. doi:10.1007/BF01234152.
URL http://dx.doi.org/10.1007/BF01234152

[20] H. Klinkrad, One year of conjunction events of ers-1 and ers-2 with objects of the usspacecom catalog, in: Second European Conference on Space Debris, Vol. 393, Darmstadt, Germany, 1997, p. 601.

[21] J. Woodburn, V. Coppola, F. Stoner, A description of filters for minimizing the time required for orbital conjunction computations, Advances in the Astronautical Sciences 135 (2) (2009) 1157–1173.

[22] L. M. Healy, Close conjunction detection on parallel computer, Journal of Guidance, Control, and Dynamics 18 (4) (1995) 824–829. doi:10.2514/3.21465.
URL http://dx.doi.org/10.2514/3.21465

[23] J. R. Alarcón Rodríguez, F. Martínez Fadrique, H. Klinkrad, Collision Risk Assessment with a 'Smart Sieve' Method, in: B. Battrick, C. Preyssi (Eds.), Joint ESA-NASA Space-Flight Safety Conference, Vol. 486 of ESA Special Publication, 2002, p. 159.

[24] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, P. Kevin Tucker, Surrogate-based analysis and optimization, Progress in Aerospace Sciences 41 (1) (2005) 1–28. doi:10.1016/j.paerosci.2005.02.001.
URL http://dx.doi.org/10.1016/j.paerosci.2005.02.001

[25] E. Iuliano, E. A. Prez, Application of Surrogate-based Global Optimization to Aerodynamic Design, 1st Edition, Springer Publishing Company, Incorporated, 2015.

[26] J. P. Boyd, Finding the zeros of a univariate equation: proxy rootfinders, chebyshev interpolation, and the companion matrix, SIAM review 55 (2) (2013) 375–396.

[27] T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun guide, accessed: 03/01/2019 (2014).
URL "http://www.chebfun.org/docs/guide/chebfun_guide.pdf"

[28] I. H. Bell, L. Bouck, B. K. Alpert, Chebtools: C++ 11 (and python) tools for working with chebyshev expansions, The Journal of Open Source Software 3 (22). doi:10.21105/joss.00569.

[29] O. Verdier, C. Swierczewski, pychebfun - python chebyshev functions, accessed: 03/01/2019 (2016).
URL https://github.com/pychebfun/pychebfun

[30] P. Dreesen, K. Batselier, B. D. Moor, Back to the roots: Polynomial system solving, linear algebra, systems theory, in: Proc 16th IFAC Symposium on System Identification (SYSID 2012), Vol. 45, Elsevier BV, 2012, pp. 1203–1208. doi:10.3182/20120711-3-be-2027.00217.
URL http://dx.doi.org/10.3182/20120711-3-BE-2027.00217

[31] C. W. Clenshaw, A. R. Curtis, A method for numerical integration on an automatic computer, Numerische Mathematik 2 (1) (1960) 197–205.

[32] J. Walter, M. Koch, G. Winkler, D. Bellot, ublas - boost basic linear algebra library, accessed: 03/01/2019 (2012).
URL https://www.boost.org/doc/libs/1_70_0/libs/numeric/ublas/doc/index.html

[33] G. Guennebaud, B. Jacob, et al., Eigen: a c++ linear algebra library, accessed 08/01/2019 (2014).
URL http://eigen.tuxfamily.org

[34] C. Sanderson, R. Curtin, Armadillo: a template-based c++ library for linear algebra, Journal of Open Source Software.

[35] H. Klinkrad, Space Debris: Models and Risk Analysis, Springer Berlin Heidelberg, 2006, pp. 13–16. doi:10.1007/3-540-37674-7.
URL http://dx.doi.org/10.1007/3-540-37674-7

[36] T. S. Kelso, Satellite orbital conjunction reports assessing threatening encounters in space(SOCRATES), accessed: 22/10/2019 (Oct. 2000).
URL http://www.celestrak.com/SOCRATES/